UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

App para asistir a pilotos y copilotos en competiciones automovilísticas

Guillermo Díaz Juan
Tutor: Francisco Jurado Monroy
Ponente: Jaime Moreno Llorena

JUNIO 2018

App para asistir a pilotos y copilotos en competiciones automovilísticas

AUTOR: Guillermo Díaz Juan
TUTOR: Francisco Jurado Monroy

Dpto. Ingeniería informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
junio de 2018

Resumen

Este Trabajo Fin de Grado trata de la realización de una aplicación Android orientada a dispositivos móviles con el objetivo de ayudar y asistir a pilotos y copilotos de competiciones automovilísticas. Principalmente esta aplicación se ha diseñado para las competiciones de rallies tanto amateur como profesionales.

En la actualidad existen pocas aplicaciones que se centren en las competiciones de rallies. La mayoría de competidores sigue utilizando un cuaderno con apuntes de unas distancias y tiempos imprecisos y suelen ser difíciles de realizar debido al movimiento que tiene el propio vehículo. Esta es la idea base a mejorar mediante la aplicación que se ha diseñado e implementado.

Previamente a la realización de este Trabajo Fin de Grado se ha realizado un estudio de las aplicaciones que existen y se ha hecho un análisis de los requisitos que un competidor de rallies requiere en una aplicación. Una vez hecho esto, mediante una estimación y siguiendo una metodología de ingeniería del software sencilla se ha desarrollado e integrado la aplicación en un tiempo adecuado.

Esta aplicación servirá para el análisis y guardado de datos de los distintos circuitos que recorran los competidores de carreras. Para analizar los datos se utilizarán coordenadas GPS para poder obtener distancias y el dibujo del trazado realizado por los usuarios, importantes para estudiar los circuitos y sus componentes o como mejorar su conducción o reglajes. Además, permite un calculo de tiempo de realización de los tramos acotados por los usuarios. Por otro lado, la aplicación contiene una manera de guardado más efectiva que un cuaderno gracias a una base de datos que contendrá la aplicación.

Estos requisitos se han implementado siguiendo estándares de Google en la estructura de los modelos, su interfaz y su codificación para una posible subida de la aplicación a Google Play.

Palabras clave

GPS, sistema operativo, Android, rally, circuito, tramo, anotación, aplicación, información, localización, analizar, sector, requisito, arquitectura, base de datos, clase, actividad, Espresso, Checkstyle.

Abstract

This Bachelor Thesis addresses to create an Android application orients to mobile devices with the target to help and assist competition races' pilots and copilots. Mainly, this application was designed for rallies.

Nowadays exist few applications for competitions of rallies. Most of competitors keep taking notes in paper to save imprecise distances and times which are difficult to write because the car is in motion. This is the main idea to improve with the application which is designed and implemented.

Previously to the development of this Bachelor Thesis, an study was made for different applications and an analysis of the requirements that a rally competitor requires in an application. Later, with a estimation of development times and a simply software engineering methodology the application has been developed and integrated in an appropriate time.

This application will make analysis and save of data from distinct tracks which users will race. To analyze data GPS points will be used to obtain distances and the route's signal followed by the users, this data is important for study of tracks and its components or improve the competitor's driving. Furthermore, the application can measure times in every section of races' sectors. Besides, a database is better saving data than a paper.

The requirements was made following Google's standards in application's structure, models, interface and coding for an a possible upload to Google Play.

Keywords

GPS, operating system, Android, rally, track, section, notation, application, information, localization, analyze, sector, requirement, architecture, database, class, activity, Espresso, Checkstyle.

Agradecimientos

Este trabajo no se podría haber hecho gracias a la gran fuerza de soportarme de mis padres, a los cuales les he dado mucho la tabarra con que no puedo perder el tiempo para estudiar y hacer prácticas de la carrera.

Sobre todo, muchísimas gracias a cada uno de mis mamos y compañeros de batallas, aventuras y plays que son Fran e Iván, sin vosotros yo creo que esta carrera nunca la hubiera terminado.

Para la realización de este TFG he de agradecer las vitales aportaciones para las imágenes a Iván y a la honra de Luis Miguel por cederme su dispositivo móvil para testar la aplicación que se ha realizado.

INDICE DE CONTENIDOS

1. Introducción **1**
1.1. Motivación 1
1.2. Objetivos 2
1.3. Organización de la memoria 3
2. Estado del arte **5**
2.1. Análisis de aplicaciones relacionadas 5
2.2. Resumen 8
3. Diseño **11**
3.1. Investigación previa 11
3.2. Metodología 11
3.3. Análisis 13
3.4. Arquitectura 17
3.5. Planificación 21
4. Desarrollo **23**
4.1. Herramientas de trabajo 23
4.2. Elementos de la aplicación 23
4.3. Funcionalidades de la aplicación 26
5. Integración, pruebas y resultados **33**
5.1. Decisiones importantes de la implementación 33
5.2. Pruebas 34
6. Conclusiones y trabajo futuro **37**
6.1. Conclusiones 37
6.2. Trabajo futuro 37
Referencias **39**
Glosario **41**

INDICE DE FIGURAS

FIGURA 3-1: METODOLOGÍA DE CASCADA…….…… .12

FIGURA 3-2: DIAGRAMA DE CASOS DE USO ……..……14

FIGURA 3-3: DIAGRAMA DE FLUJO DE DATOS SIMPLIFICADO ……..17

FIGURA 3-4: ARQUITECTURA MODELO-VISTA-CONTROLADOR…… .18

FIGURA 3-5: DIAGRAMA DE CLASES EN LA CAPA DE MODELO……19

FIGURA 3-6: DIAGRAMA DE ENTIDAD-RELACION DE LA BASE DE DATOS……20

FIGURA 3-7: DIAGRAMA DE GANTT……..……21

FIGURA 4-1: CICLO DE VIDA DE UNA ACTIVIDAD……24

FIGURA 4-2: LISTADO DE CIRCUITOS - AÑADIR CIRCUITO……26

FIGURA 4-3: LISTADO DE CIRCUITOS - BORRAR CIRCUITO…….. 27

FIGURA 4-4:LISTADO DE SECTORES……..……28

FIGURA 4-5:LISTADO DE TRAMOS……..…….. 28

FIGURA 4-6: ACTIVIDAD: REGISTRAR TRAMOS……..…….. 29

FIGURA 4-7: ACTIVIDAD: AÑADIR INFORMACIÓN EXTRA……..……31

FIGURA 4-8: ACTIVIDAD: MOSTAR INFORMACIÓN EN CARRERA.. ……..……32

FIGURA 5-1: COMO HALLAR EL ANGULO ENTRE DOS RECTA…..……..……33

FIGURA 5-2:USO DE CHECKSTYLE…….. ……..……35

INDICE DE TABLAS

TABLA 2-1: TABLA DE RESUMEN DE APLICACIONES SEGÚN SUS CARACTERISTICAS.9

TABLA 2-2: TABLA DE RESUMEN DE APLICACIONES SEGUN SUS RECURSOS …….10

1.Introducción

1.1.Motivación

Actualmente en las carreras se busca tener referencias y mediciones de muchos tipos de datos: velocidad, temperatura, aerodinámica, etc. Muchas de estas mediciones se hacen con aparatos específicos y muy caros. Aunque gracias a la tecnología de hoy en día, un simple smartphone puede dar datos acera de la velocidad o de los puntos GPS por los cuales ha pasado un vehículo. Todas estas mediciones son muy importantes para los competidores ya que se emplean para adaptar los vehículos y mejorar marcas ya sean personales o competitivas.

La finalidad del proyecto es la de asistir a pilotos y copilotos de competiciones automovilísticas, ya sea en circuito cerrado o en circuito abierto mediante una aplicación para el sistema operativo Android.

Este proyecto se centrará principalmente en la prueba deportiva del rally. Para ello vamos a explicar un poco cómo se compite en esta modalidad de carreras y las funciones de cada uno de los miembros del vehículo.

Para competir, primero se realiza un reconocimiento a siguiendo las leyes de tráfico que existan durante el recorrido, donde los pilotos aportan datos de estimación sobre: distancias, velocidad máxima alcanzada, dirección y cantidad de giro realizado con el volante u objetos del paisaje que les haga adoptar una acción determinada. Después, se realiza un entrenamiento del circuito cómo si fuera una carrera para matizar los valores dados en el reconocimiento. Toda esta información la apunta el copiloto en un cuaderno para después “cantársela” durante la carrera al piloto.

Después se realizan etapas cronometradas para escoger en qué posición salir en la carrera, buscando a su favor el estado del camino y el tiempo meteorológico.

Por último, se corre la carrera a contrarreloj, solamente un coche cada vez, y gana quien hay recorrido todos los tramos de la competición en menos tiempo, de media. [1]

Las anotaciones de los pilotos y copilotos deben ser fiables para que durante la carrera los pilotos puedan realizar el mejor trazado posible y de una manera adecuada. Estas anotaciones son muy subjetivas, puesto que hay pilotos que prefieren tener apuntes de kilómetros por hora y otros de marchas y revoluciones para entender la velocidad, o un numero relacionado con las horas del reloj o una numeración señalada en el volante para saber cuanto giro se va a realizar.

Puesto que existen pocas aplicaciones que se conozcan y los competidores aún toman apuntes en papel. Con esta aplicación se pretende sustituir el cuaderno por un dispositivo móvil con un tamaño de pantalla grande (smartphone o Tablet).

Su función será ir guardando información y mostrar esta al piloto y/o copiloto de una manera familiar, para que tengan una referencia más a la hora de participar en una competición. Algunos aspectos destacables de esta aplicación será que se podrá utilizarse sin conexión a internet, se eliminarán las notas manuscritas en papel, y se proveerá de un cálculo más aproximado tanto en tiempos como en distancias.

1.2.Objetivos

El objetivo principal de la aplicación será asistir a participantes de carreras automovilísticas, haciendo una sustitución del papel y el boli o la ayuda de varios aparatos distintos y juntar todo esto en una aplicación Android para smartphones de pantalla grande y tablets.

La aplicación constará con dos modos de utilización: guardar información y mostrar dicha información recogida previamente, con unos datos que puedan ser reconocibles para los competidores.

Para poder dar soporte a los usuarios se han planteado 3 escenarios que seguirán el siguiente orden.

Primero, la adquisición de datos por parte del dispositivo móvil. La aplicación se encargará de guardar los datos relacionados con los tiempos, distancias recorridas y localizaciones a través del GPS que integre dicho dispositivo o los datos móviles que pueda utilizar.

Segundo, el análisis de los datos recopilados previamente. El dispositivo analizará cada una de las coordenadas que haya almacenado y mostrará de manera intuitiva a los usuarios: qué indicaciones o caminos han seguido mediante imágenes y una estimación de velocidad, tiempo y distancia que ha tenido el usuario durante su recorrido.

Por último, la muestra de los datos analizados al usuario de manera automática y personalizada. Los usuarios podrán añadir información adicional a utilizar para cada una de las indicaciones analizadas y se mostrarán los datos de manera manual, según el usuario quiera.

La finalidad del proyecto es encontrar una manera más cómoda y descargar el trabajo de los competidores para poder centrarse más en el circuito, puesto que habrá sucesos que ocurren muy rápido y quizás haya puntos donde el usuario no pueda guardar toda la información necesaria para la competición si tuviera que registrar él mismo en un papel.

Además, una posibilidad para la mejora de la experiencia de los usuarios es utilizar los datos que guarde la aplicación para realizar telemetrías y analizar donde están los puntos fuertes de sus automóviles o ver en que tramos o sectores hay que mejorar tiempos.

1.3.Organización de la memoria

La memoria del presente Trabajo de Fin de Grado que trata sobre una aplicación para asistir a pilotos y/o copilotos de competiciones automovilísticas desarrollada en Android, constará de los siguientes apartados:

- Capítulo 1: Motivación, objetivos y organización de la memoria. Enumeración de motivos y objetivos de este TFG y cómo será su organización a lo largo de este documento.

- Capítulo 2: Estado del arte. Análisis de otras aplicaciones existentes.

- Capítulo 3: Diseño. Planificación de cómo desarrollar la aplicación.

- Capítulo 4: Desarrollo. Explicación de las acciones realizadas y componentes de la aplicación.

- Capítulo 5: Integración, pruebas y resultados.

- Capítulo 6: Conclusiones y trabajo futuro.

2.Estado del arte

En este capítulo se analizarán otras posibles aplicaciones que tengan relación con el presente Trabajo Final de Grado.

Para ello, se han examinado las disposiciones de varias aplicaciones, buscando los matices comunes entre ellos, tanto como puntos buenos y puntos malos, para así ayudar a la creación de este Trabajo de Fin de Grado.

2.1.Análisis de aplicaciones relacionadas

Existen algunas aplicaciones destinadas a dar soporte a participantes en competiciones automovilísticas deportivas, sin embargo, no todas cubren el mismo abanico de posibilidades ni funcionalidades. En esta sección se analizarán las principales características tanto a tener en cuenta como a evitar en la aplicación que se diseñará a lo largo del presente TFG.

2.1.1.RallyGDP (TSD Rally Computer)

Esta aplicación gratuita para Android comienza con un menú sencillo de tres opciones: realizar un rally sin un archivo previo, utilizar un archivo de rally guardado anteriormente como un fichero de extensión propia, e importar uno que haya en memoria.

Esta app utiliza el GPS del dispositivo y permite guardar los datos manualmente de cuál es la distancia que se va a recorrer y en cuanto tiempo se quiere realizar. Además utiliza mapas generados por Google Maps, tiene un modo de utilizar botones en el que es necesario mantener el botón pulsado para que ejecute la acción correspondiente.

El principal defecto que se encuentra a esta aplicación es que todo es manual, debe introducir uno mismo los datos anteriormente.

Un buen método de pulsar la pantalla es que para hacer click en un botón hay que mantener pulsada la pantalla. Otro dato relevantes es si se quiere abrir la aplicación con datos guardados previamente o empezar de cero el recopilado de datos.

2.1.2.Copilote Master

Esta aplicación gratuita para Android es muy sencilla, con una interfaz de usuario fácil gracias a su apartado HELP. Se basa en varios apartados: checkpoint, mapa, cronometro, tiempo atmosférico, horario y contactos rápidos.

La utilización de GPS se realiza mediante el API de Google Maps, donde además muestra a qué velocidad se mueve el usuario en el momento y cuál es el tiempo actual en todo momento.

Detalles a tener en cuenta es el cronometro que guarda tiempos de manera particionada, pero no hay nada relativo a distancias. Otro dato interesante que aporta es el tiempo atmosférico de un sitio, ya que aporta gran información a los usuarios.

2.1.3.RallyTripmeter

Esta aplicación de Android es de descarga gratuita, pero para poder usar su contenido hay que pagar el modo premium.

Consta de varios apartados: “Recce” para saber qué notas pueden tomarse actualmente relativas a velocidad, tiempo y distancia; “Liaison” que sirve para marcar un tiempo límite para un cierto tramo, donde muestra cuánto queda para empezar y el tiempo límite una vez llegado; y por último, “Race” que sirve para saber de media cuál es la máxima velocidad, cuál es la velocidad media, cuántos kilómetros se han recorrido y la velocidad actual.

Los puntos fuertes de la aplicación son el mostrar las diferentes partes de la carreras de rallies (“Recce”, “Liaison” y “Race”), especializar en cada uno de los apartados la manera de mostrar los datos de velocidades y distancias recorridas, y por último, el tener apartados en el menú para ver el tiempo atmosférico y otro para etapas especiales.

Por otro lado, no hay manera de ver que tipo de recorrido se ha seguido, y las estimaciones tienes que marcarlas antes de empezar a conducir, ya que no guarda los datos obtenidos anteriormente.

2.1.4. Rally Navigator

Es una aplicación desarrollada para el sistema operativo Android de descarga gratuita.

Es la más sencilla de utilizar, comenzando por un fragmento que pregunta el tipo de GPS se va a utilizar (interno o externo al dispositivo móvil). A partir de aquí aparecen una lista de circuitos con un botón de añadir un nuevo circuito o sector. Si empezamos una carrera o sector nuevo, aparece una anotación en blanco, y presionando un botón se empieza un nuevo tramo. Durante la realización de un tramo o después de haberlo realizado se pueden añadir anotaciones tanto escritas mediante el teclado, mediante un audio de la grabadora o añadir una imagen.

Una de las partes más interesantes de esta aplicación es que se pueden sincronizar los libros de anotaciones por versión web, versión móvil o ambas maneras. También tiene como gran cualidad el hecho de mostrar que tipos de movimiento se están realizando y el poder añadir varios tipos de anotaciones en cualquier momento.

Por el contrario, no te deja salir de la aplicación a no ser que no se cierre desde el menú de aplicaciones o forzando la salida.

2.1.5.Rabbit 2.0

Esta aplicación gratuita para Android algo difícil de utilizar ya que los métodos de introducción de datos son manuales. Se pueden llevar dos medidores de velocidad o de kilometraje para poder ir realizando apuntes a mano, pero no es fácil para el usuario tener que estar cambiando de pantallas para poder ir mirando los tiempos actuales o poder realizar ajustes de medición.

Es una aplicación con una interfaz gráfica muy vistosa, pero con bastantes partes de la pantalla inútiles o que contienen apuntes con falta de sentido.

Partes muy sensatas es que suene un pitido cada cierto tiempo para poder saber cuánto tiempo está pasando, además de no finalizar la aplicación cuando se sale de ella de manera que no sea mediante un botón que contiene el menú.

2.1.6.Belmont Rallie

Esta aplicación es para Android y gratuita. Es muy simple, solo dos actividades: mostrar la distancia y la velocidad a que se está yendo en el momento, y las opciones de la app.

En la pantalla que contiene los contadores de la distancia y el tiempo actuales aparece una brújula y el tiempo total desde que se inicia el primer cronómetro.

La flaqueza de esta aplicación es que no guarda más datos que el tiempo total, pero no se muestran distancias para las notaciones o no se muestra una dirección o cuantas veces se ha reiniciado el cronómetro de tiempos (para contar tramos, por ejemplo).

2.1.7.Harry's LapTimer Grand Prix

Esta aplicación es de pago para iOS. Es una aplicación para circuitos cerrados, donde se piden datos de un usuario para saber qué coche utilizará y sus características, como el motor. Después se añade un circuito de carreras existente en la aplicación o subidos a internet por otros usuarios. Se pueden buscar por tipo de circuito o de competición.

A la hora de competir, se muestran dos relojes, uno con el tiempo de la vuelta actual y otro con la diferencia de tiempo frente a la mejor vuelta guardada del circuito. Al finalizar te muestra las estadísticas de la carrera.

2.1.8.Ultimate Rally CoDriver

Esta aplicación libre o de pago en iOS. Es compatible para iPhone, iPad e iPod touch. Se trata de una aplicación que recibe datos guardados previamente en algún servidor y que los usuarios de la app se descargan dependiendo de que circuito vayan a hacer y que localización tengan.

La aplicación tiene diferentes secciones separadas por las etapas de una carrera de rallies: “Pre-Rally”, “Recce”, “Rally”. En la actividad principal se puede escoger una de las etapas de carrera, acceder a los ajustes de la aplicación y de los rallies o acceder a un calendario

de próximos rallies y ver los rallies archivados en la memoria del dispositivo donde corre la aplicación.

En la sección de “Pre-Rally” se pueden acceder a los datos del itinerario de viaje, links para ver el tiempo y las regulaciones de tráfico. En “Recce”, se pueden ver los detalles de las etapas de una carrera, marcar tiempos predefinidos y ver tiempos realizados realmente durante la fase de simulación de carrera. Por último, en “Rally” se pueden ver tiempos de los tramos en las notas, la lista de acciones mecánicas de que está todo correcto, un calculador de gasolina, un cronómetro y características de la etapa.

Los puntos fuertes son que la app permite editar la información servida, el tener las etapas del rally separadas por secciones y el análisis de los tiempos estimados frente a los tiempos realizados realmente (función muy importante para los pilotos y copilotos de rallies).
Por el contrario, su flaqueza es que tiene muchísima información que debe ser introducida a mano y los datos se verían muy pequeños si el dispositivo no fuera un iPad.

2.2.Resumen

En la tabla 2-1 se hace un pequeño repaso de los sistemas operativos en los cuales se pueden utilizar las aplicaciones además de saber si son de pago y los dispositivos en los que se pueden utilizar.

En el caso de Android, casi todas las versiones son 4.x, siendo esta la versión ahora mismo más extendida. Por otro lado, en iOS, todos los dispositivos tienen que ser dispositivos móviles, y versión 8.0 o superior.

En la tabla 2-2 se puede ver un resumen de los recursos que usan las aplicaciones analizadas previamente. Lo más utilizado es el GPS para tener una velocidad y una distancia a analizar. Otra característica que comparten varias aplicaciones es la de guardar datos del circuito en la memoria del dispositivo.

Muchas aplicaciones rehúsan el utilizar símbolos para las notas de los pilotos o guardar la información de manera más automatizada.

Características App	Android/iOS	Gratuita/ pago	Dispositivos compatibles
RallyGDP	Android	Gratuita y pago	- Android 4.0 y versiones posteriores.
Copilote Master	Android/iOS	Pago	- Android 4.0 y versiones posteriores. - iOS 8.0 o superior. iPad, iPhone y iPod touch
RallyTripmeter	Android/iOS	Gratuita	- Android 4.1 y versiones posteriores. - iOS 9.0 o posterior. Compatible con iPhone, iPad y iPod touch.
Rally Navigator	Android/iOS	Gratuita y pago	- Android 4.0 y versiones posteriores. - iOS 8.0 o superior. iPad, iPhone y iPod touch.
Rabbit 2.0	Android/iOS	Gratuita	- Android 4.2 y versiones posteriores. - iOS 8.0 o superior. iPad, iPhone y iPod touch.
Belmont Rallie	Android	Gratuita	- Android 4.4 y posterior.
Harry's LapTimer Grand Prix	Android/iOS	Pago	- Android 4.0 y versiones posteriores. - iOS 9.3 o posterior. Compatible con iPhone, iPad y iPod touch.

Tabla 2-1: Tabla de resumen de aplicaciones según sus características

\Recursos App	GPS	Mapa	Guardado Automático	Guardar circuito/s	Imágenes auxiliares	Añadir información al tramo
RallyGDP	Si	Si	No	Si	No	No
Copilote Master	Si	Si	No	Si	No	No
RallyTripmeter	Si	No	No	No	No	No
Rally Navigator	Si	No	Si	Si	Si	Si
Rabbit 2.0	Si	No	No	Si	No	No
Belmont Rallie	Si	No	No	No	No	No
Harry's LapTimer Grand Prix	Si	Si	No	Si	Si	No

Tabla 2-2: Tabla de resumen de aplicaciones según sus recursos

3.Diseño

Para poder realizar un diseño, previamente hay que estudiar el sistema operativo donde se ejecutará la aplicación, que versiones de dicho S.O. pueden ser compatibles con el software y que flujo de trabajo seguir. Después, hay que especificar de manera completa cual será sus flujos de comportamientos del software, de datos y de control. A continuación, se mostrará un análisis del problema y los requisitos funcionales y no funcional que debe cumplir el software creado.

3.1.Investigación previa

En este apartado se plantea que sistema operativo escoger y serán explicadas las razones de la elección de dicho S.O. escogido. De aquí aparecerán los requisitos no funcionales que deberá tener la aplicación a diseñar.

3.1.1.Sistema Operativo

El S.O. donde se instalará el software será Android, por ser un sistema operativo gratuito, multiplataforma y libre, haciendo que Android sea la elección para programar este TFG.

Android no es un lenguaje de programación, está basado en Java, del cual se aprovechará la ventaja que tiene para manejar la memoria de manera eficiente [2], además de no tener tantos errores en tiempo de ejecución, muy importante para cualquier aplicación.

3.1.2.Versiones y dispositivos compatibles

Las versiones más utilizadas de Android a finales de 2017 son: Marshmallow, 6.0 (29.7%); Lollipop, 5.0 (26.3%); y Nougat, 7.0 (23.3%). [3] Jelly Bean, con su versión 4.3, ha sido la versión mínima elegida ya que pueden realizarse pruebas de integración sobre la interfaz gracias a la librería Espresso y la librería uiautomator:

```
androidTestImplementation
'com.android.support.test.uiautomator:uiautomator-v18:2.1.3'
```

Para utilizar la localización y la geolocalización del dispositivo es compatible desde la versión 1.0, así que para las versiones de Android que contenga la aplicación de este TFG.

Para los dispositivos, principalmente la aplicación está destinada a Smartphones con tamaños de pantalla grande y Tablets con GPS. Para una mejor resolución del GPS, el dispositivo puede apoyarse en los datos que se obtengan mediante Wifi o itinerancia de datos móviles.

3.2.Metodología

En este apartado se explicarán el conjunto de pasos para formalizar el desarrollo de este proyecto. También se explicará cómo se realizan estos métodos y su orden de realización.

3.2.1.Ciclo de vida

Un ciclo de vida es el orden en el cual se van a realizar los pasos del desarrollo del proyecto y las transiciones que hay que hacer entre cada uno de ellos.

En este proyecto se ha decidido seguir el ciclo de vida en cascada pura. Los pasos a seguir son muy definidos y para pasar de una etapa a otra del desarrollo del software se deberá realizar una revisión. [4]

El ciclo de cascada pura hace que se desarrolle el proyecto con una velocidad adecuada al tiempo que hay para la realización de este TFG, teniendo en cuenta que solo lo realiza una persona.

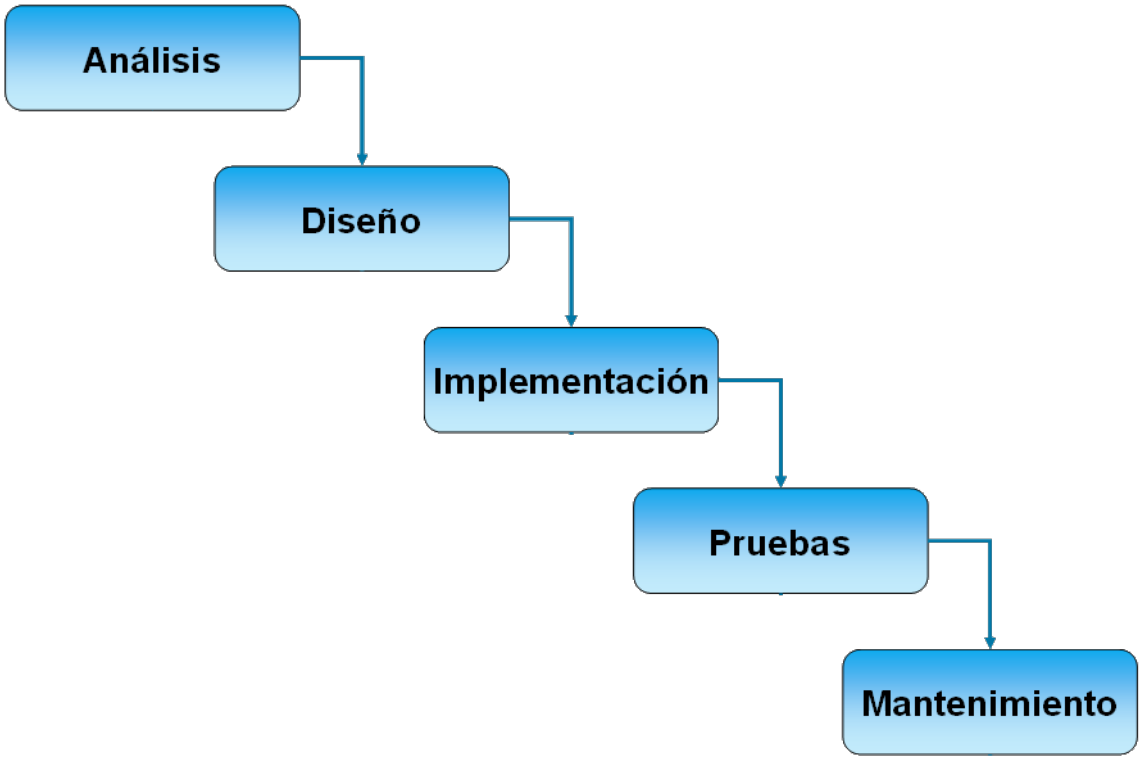

Figura 3-1: Metodología de cascada. Recuperado de [5]

- **Planificación**. Es la parte más importante para poder estimar el tiempo que conllevará cada fase de desarrollo del proyecto y cuando podría terminarse este.

- **Análisis**. Este paso está dedicado al entendimiento del problema a resolver y los requisitos que debe cumplir la aplicación, ya sea de manera funcional para el usuario y de manera no funcional hacia el comportamiento del dispositivo.

- **Diseño**. En este método se realizarán planteamientos de cómo será la interfaz de las distintas actividades sobre la pantalla del dispositivo móvil, su navegación y comunicación entre ellas. Además, se explicará la estructura que tendrá la base de datos que contenga la información relevante para el usuario.

- **Codificación o implementación**. Es el punto más difícil de estimar en tiempo, ya que es el más largo, lo que conllevaría el paso donde más riesgos puede sufrir. Durante la realización de esta etapa se procederá a realizar pruebas al más bajo nivel para la tener una perspectiva de los datos que se van a manejar e incluso, pruebas unitarias.

- **Pruebas**. En este paso se realizarán pruebas por cada uno de los módulos que contenga el código realizado, pruebas de caja negra, y además se realizarán pruebas para contemplar el correcto funcionamiento entre la comunicación de los módulos del proyecto, pruebas de caja blanca. Además, se procederá a utilizar pruebas de interacción de usuario. [6]

- **Mantenimiento**. Esta fase, aunque no realizada en este proyecto, es de las más importantes, puesto que si hubiera errores no detectados durante las pruebas, habría que solventarlos. Si el mantenimiento se viera como una tarea muy ardua se podría plantear volver a realizar el ciclo de vida completo para poder evitar tener muchos más errores en la entrega final del proyecto.

3.3.Análisis

3.3.1.Herramientas utilizadas

Se han utilizado el programa de diseño UML, ArgoUML, para el diagrama de casos de uso, y el diagrama de clases que se enseñará más adelante. Esta aplicación esta programada en Java, siendo posible utilizarlo por cualquier plataforma que soporte Java. [7]

3.3.2.Análisis del problema

Explicado en el apartado 1.1, la motivación de este TFG es crear una aplicación de asistencia a pilotos y copilotos de carreras, donde sea la propia aplicación la que guarde los datos precisos para poder usarse durante una competición. Además, la aplicación ha de facilitar la información de manera sencilla, tal que cualquier usuario pueda comprenderla.

El problema a solucionar se da en el momento en el que se está en una carrera de rallies, por ejemplo, anotar los datos “cantados” por el conductor y ver estos durante la carrera es una tarea bastante ardua, y la solución a dar es que el software guarde datos a partir de una fácil actuación del piloto o copiloto, y, que a la hora de mostrar los datos, se muestren de manera automática y con visual clara.

3.3.3.Requisitos funcionales

Los requisitos funcionales los abordaremos a partir de un diagrama de casos de usos, donde se capturan todos los requisitos funcionales que debe tener el sistema. Estos se verán representados en la figura 3-2

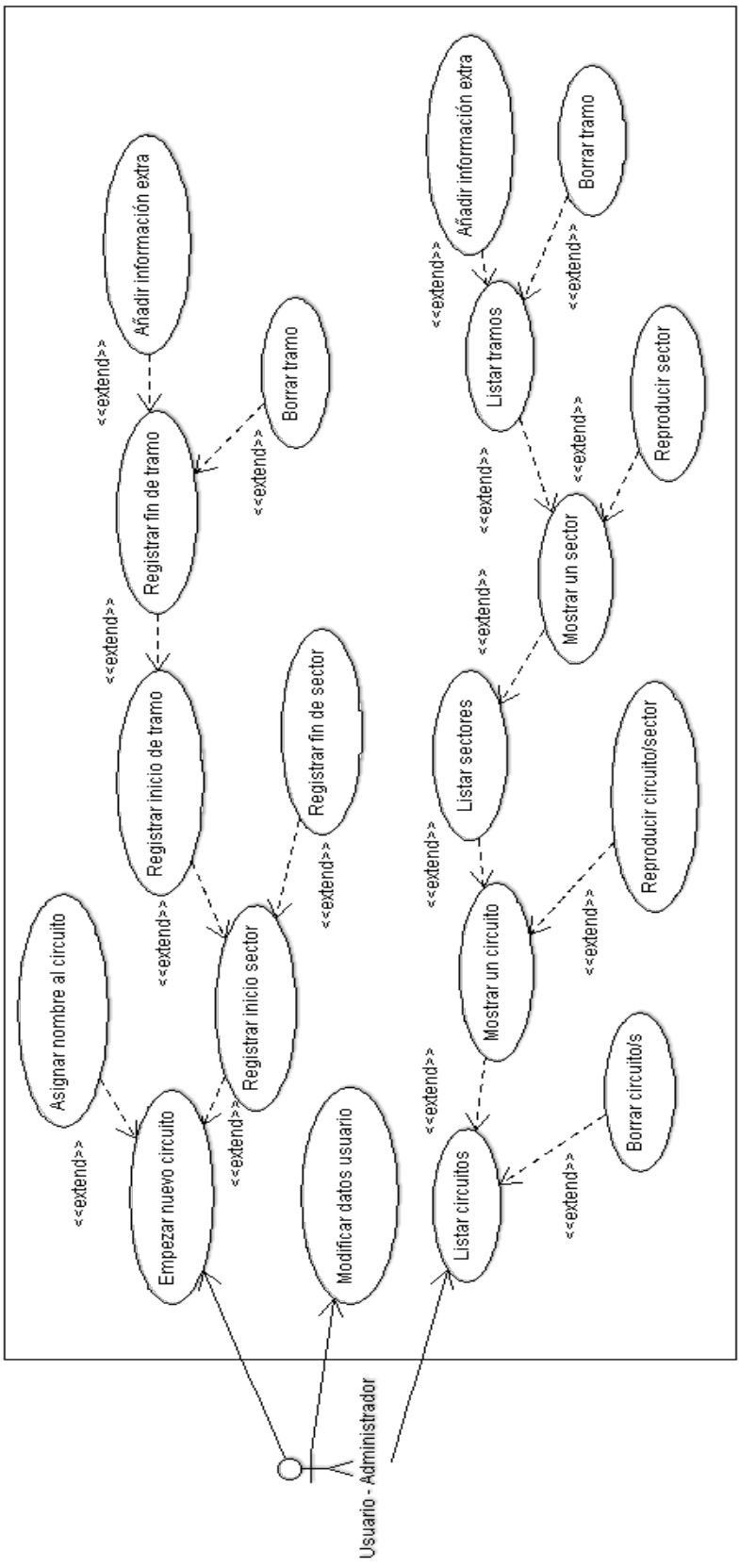

Figura 3-2: Diagrama de casos de uso

RF - 1. Empezar nuevo circuito.
El usuario pretende almacenar información de un circuito, para posteriormente que pueda ver la información de la composición de este.

RF - 2. Asignar nombre al circuito.
Al crear un circuito, es necesario asignarle un nombre único en el dispositivo para la posterior búsqueda de datos que se contenga en el dispositivo.

RF - 3. Registrar inicio de sector.
El usuario, dentro del espacio de una carrera puede registrar el inicio de un sector para guardar información sobre este. No será necesario que el usuario añada un nombre al sector.

RF - 4. Registrar fin de sector.
Cuando el usuario haya terminado de guardar información (tramos) dentro del sector, este deberá indicarlo al dispositivo.

RF - 5. Registrar inicio de tramo.
Durante un sector, el usuario activa el inicio de registro de información en el dispositivo a partir de una actuación sobre el software.

RF - 6. Registrar fin de tramo.
Cuando se quiere terminar de recolectar datos sobre un tramo, el usuario realizará el mismo gesto sobre el dispositivo para finalizar la recogida de datos sobre el tramo actual y comenzará automáticamente un nuevo tramo.

RF - 7. Añadir información extra a tramo.
Durante la recogida de datos en un tramo, el usuario podrá añadir información extra sobre el tramo actual.

RF - 8. Borrar tramo.
El usuario podrá borrar uno o varios tramos, según desee.

RF - 9. Modificar datos de usuario.
El usuario podrá cambiar su nombre o datos relacionados con él, transformación de velocidades o coche actual.

RF - 10. Listar circuitos.
Al iniciar la aplicación y seleccionar un usuario, se listarán los circuitos asociados al usuario.

RF - 11. Borrar circuitos.
Durante el listado de circuitos, el usuario podrá eliminar uno o varios circuitos.

RF - 12. Mostrar circuito.
El usuario seleccionará un circuito de la lista y se mostrarán los datos que contiene agrupados en la suma de los sectores que contenga.

RF - 13. Listar sectores.
Una vez se esté mostrando un circuito, el usuario podrá acceder al listado de sectores que contiene el circuito, cada uno con su información correspondiente.

RF - 14. Reproducir circuito/sector.
Mientras se está mostrando un circuito o un sector en la aplicación, el usuario puede reproducir este, en el que se mostraran sus tramos correspondientes de manera que vaya pasando por ellos.

RF - 15. Mostrar un sector.
A partir de la lista de tramos, el usuario podrá acceder la información específica de un tramo.

RF - 16. Listar tramos.
Mientras se muestra un sector, el usuario puede acceder al listado de tramos de este, donde se muestran de forma resumida los datos de cada tramo.

RF - 17. Añadir información extra.
Durante el listado de tramos de un sector, el usuario puede añadir información extra a cualquier tramo para ayuda de mostrado de datos.

RF - 18. Borrar tramo.
Durante el listado de tramos de un sector, el usuario podrá borrar uno o varios tramos del sector seleccionado.

3.3.4.Requisitos no funcionales

Los requisitos no funcionales serán descritos a continuación. Estos requisitos los debe realizar el software sin que el usuario intervenga en ellos y son imprescindibles para la aplicación, ya que denotan aspectos importantes del software a realizar.

RNF - 1. Compatibilidad de versión Android.
Esta aplicación deberá ser compatible para la versión de Android 4.0.3 hasta la versión 8.1 que es la más actual. [8]

RNF - 2. Almacenamiento de datos.
Los datos deben estar guardados en el dispositivo, donde se guardaran en logs y/o en bases de datos, para dar fluidez al dispositivo.

RNF - 3. Interfaz intuitiva de usuario.
La aplicación seguirá los estándares de Material Design que proporciona Android, haciendo que la parte visual de la aplicación sea familiar para el usuario.

RNF - 4. Conexión GPS.
Es fundamental la conexión de GPS para el software de este TFG, puesto que con esta conexión se podrá realizar un seguimiento y la recogida de información que necesitará el usuario de manera simplificada.

3.3.5.Flujo de datos

La aplicación seguirá un flujo representado en la figura 3-3. Se va a componer de la base de datos, la propia aplicación y el usuario que pedirá cierta información o la añadirá de manera manual. A continuación se muestra una imagen simplificada del flujo de datos con un listado de carreras y el añadido de datos a un tramo.

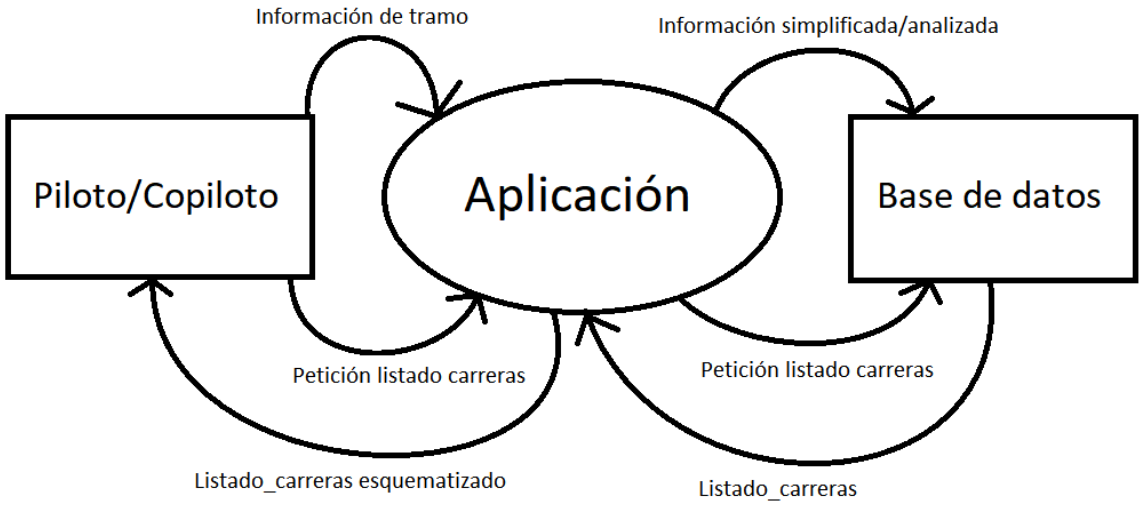

Figura 3-3: Diagrama de flujo de datos simplificado

3.4.Arquitectura

En este apartado se van a detallar las arquitecturas que se diseñarán para la aplicación. Los patrones a diseñar serán las interfaces gráficas y el manejo de datos que muestran y controlan, la base de datos y las clases que se van a utilizar. Para la explicación de estos diseños se aportarán diagramas e imágenes y la relevancia de haber escogido este tipo de arquitecturas.

3.4.1.Modelo-Vista-Controlador

El patrón utilizado para el manejo de datos, las interfaces gráficas y la lógica ha sido MVC. Esto hace que la división en módulos de la aplicación sea más efectiva a la hora de

poder reutilizar código. Esta arquitectura es muy utilizada ya sea para programas web cómo para aplicaciones nativas, como es en este caso. [9]

La capa de la vista es la que se encarga de la visualización de los datos de una manera adecuada para que el usuario pueda interactuar con la aplicación, llamado también interfaz de usuario.Además, es donde mediante acciones (ya sea presionar un botón o deslizar el dedo) envía una notificación de la acción realizada hacia el controlador.

El controlador es la capa que hace de puente entre la vista y el modelo. Al recibir la notificación de la realización de una acción por parte del usuario, accede a los datos que contiene el módulo modelo, ya sea para obtener información o modificarla. Además en esta capa se introduce la lógica de cómo transformar datos, cómo por ejemplo tener que calcular la velocidad media si no se obtiene la velocidad desde el modulo Location de Android.

Por último, está la capa de modelo que contiene toda la información necesaria referente a la aplicación, dando a las capas de vista y controlador la información que van a mostrar y manejar. Para ello, esta capa se comunica con la base de datos.

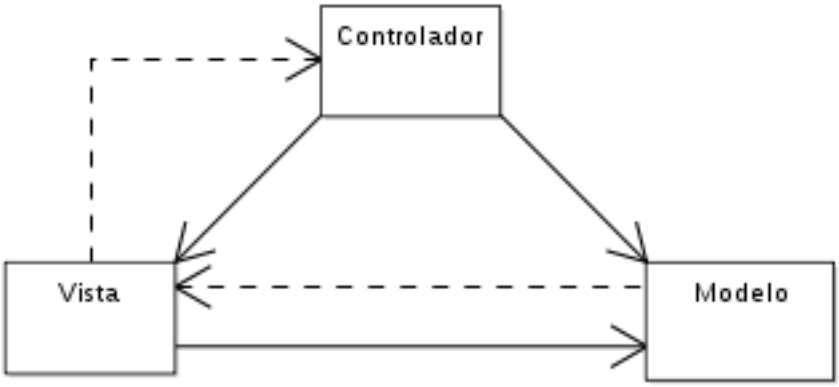

Figura 3-4: Arquitectura Modelo-Vista-Controlador. Recuperado de [10]

3.4.1.1.Clases VO (Modelo)

A la hora de implementar la capa del modelo, se han utilizado las siguientes clases que implementan el patrón Value Object (VO)[11]. En el diagrama de clases representado en la figura 3-5 se llegan a figurar los datos que contendrán algunas de las tablas de la base de datos. En este diagrama de clases se obvian los métodos get y set, que serán métodos públicos.

3.4.2.Base de datos

La base de datos que utilizará la aplicación será una base de datos relacional, donde se usará lenguaje SQL sobre el sistema de gestión de bases de datos de Android SQLite. Las tablas que contiene son:

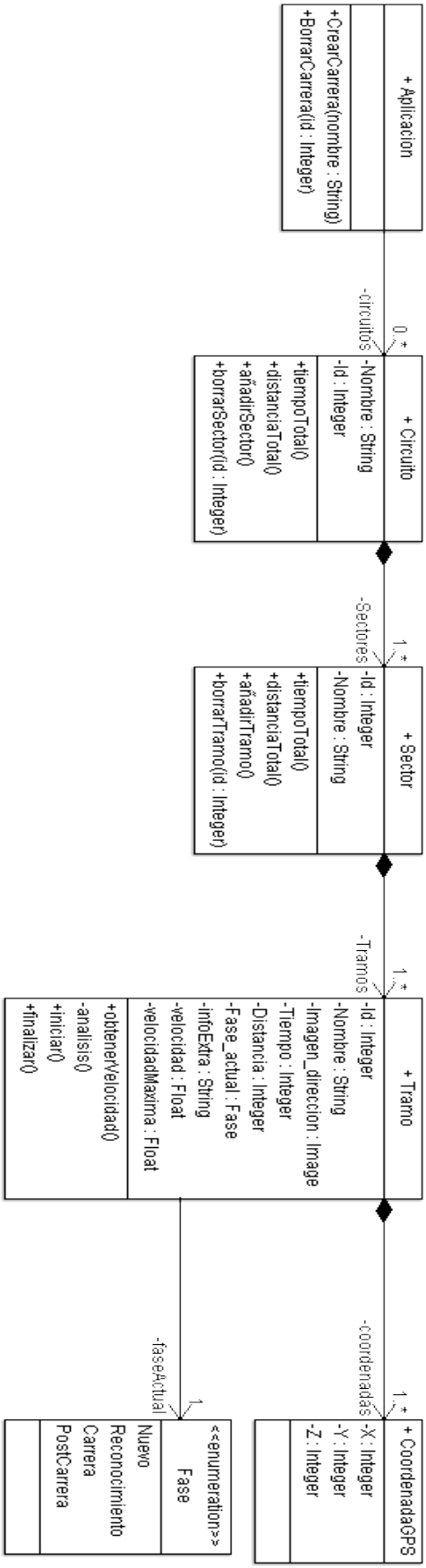

Figura 3-5: Diagrama de clases en la capa de modelo

- **Usuarios.** Esta tabla almacena los usuarios que contiene la aplicación. Estos usuarios no son compartidos entre dispositivos. Los datos relativos a los usuarios serán: identificador, nombre y contraseña.

- **Circuitos**. Los datos que se guardan en esta tabla son: identificador, nombre del circuito, identificador del usuario e identificadores de los sectores que contiene.

- **Sectores**. Los campos relativos a esta tabla son: identificador, identificador del circuito al que pertenece y identificadores de los tramos que contiene.

- **Tramos**. Esta tabla es la más extensa, por toda la información necesaria que contiene. Sus campos son: identificador, imagen de la figura que realiza el trazado, tiempo de realizarlo, distancia recorrida, fase del tramo, velocidad media, velocidad máxima y una lista de identificadores de imágenes de información extra.

Los datos todos los datos son de tipo String y números, haciendo que la base de datos no ocupe tanto espacio en la memoria del dispositivo. En la figura 3-6 se muestra un esquema relacional de cómo serán las tablas y los valores.

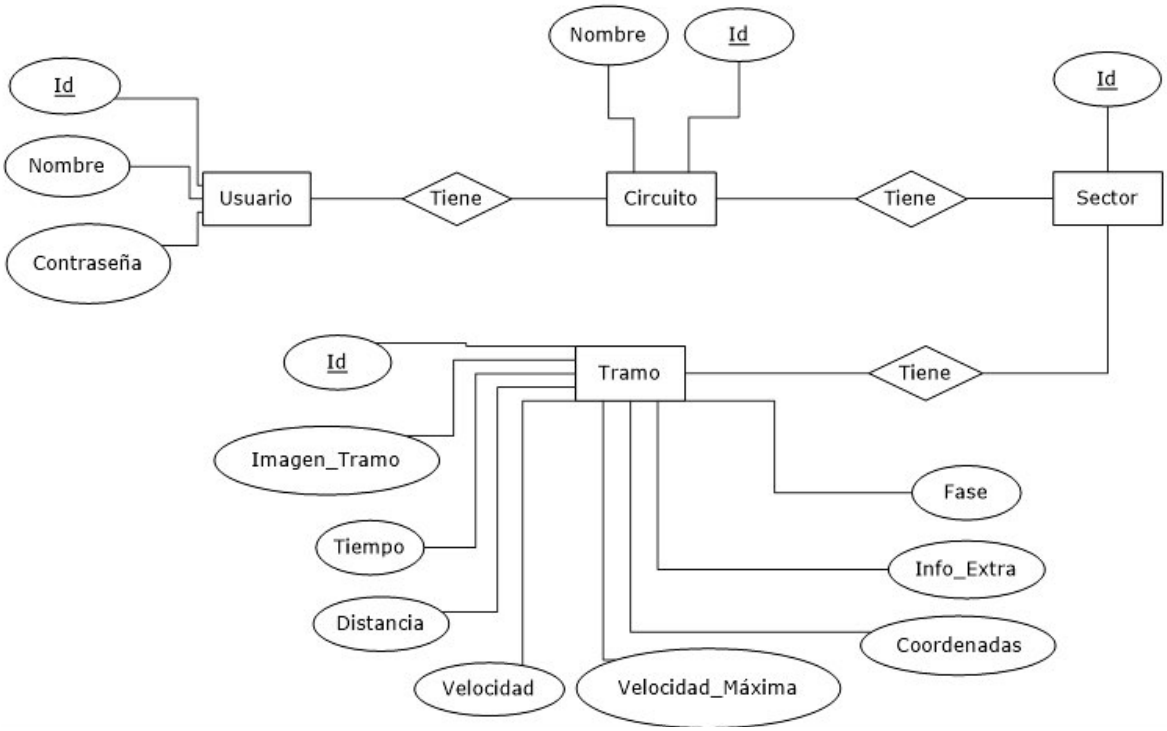

Figura 3-6: Diagrama de entidad-relación de la base de datos

3.5.Planificación

La planificación es una etapa donde se hace una estimación del tiempo de la realización del proyecto, representada en la figura 3-7. Para la plantear los tiempos de realización del proyecto se ha utilizado un diagrama de Gantt, dividido por semanas.

En este caso, al tener una fecha límite, hay que priorizar mucho los riesgos que pueden hacer que una fase del desarrollo puedan hacer retrasar el resto de partes del proyecto. Al ser un proyecto hecho en fases muy dependientes unas de otras, un retraso en una etapa de este proyecto puede retrasar todo, llegando a hacer que en la fecha límite haya falta de requisitos que serán analizados en la sección 3.3.3 y 3.3.4.

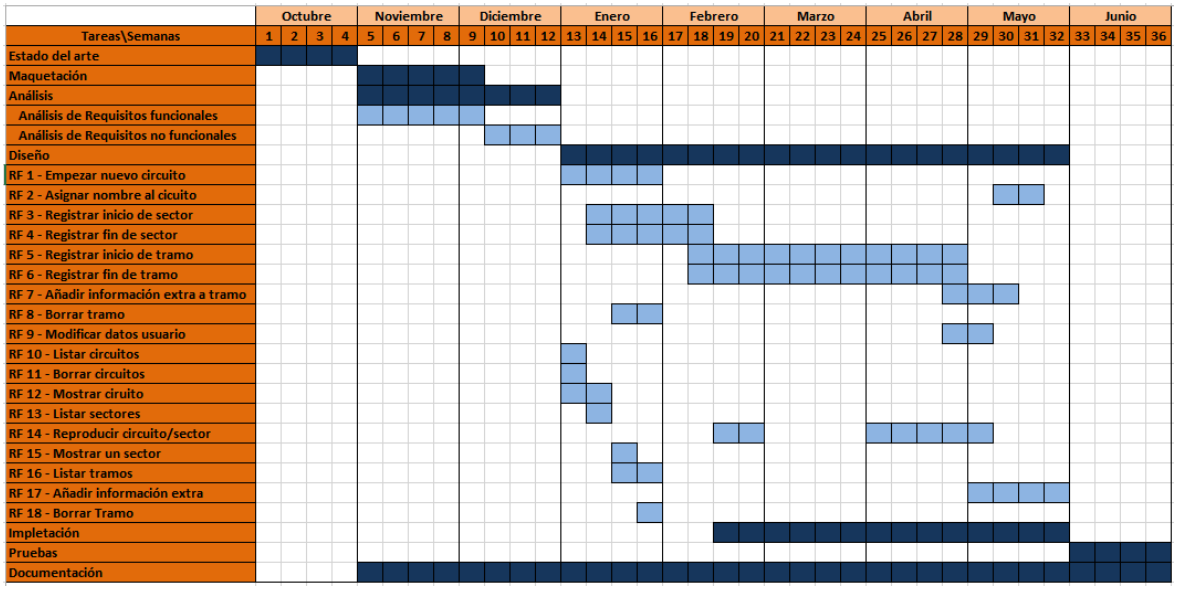

Figura 3-7: Diagrama de Gantt

4.Desarrollo

La etapa de desarrollo trata de qué soluciones se han dado a los requisitos funcionales y no funcionales de la aplicación. En esta sección se detallarán:

- Herramientas de trabajo utilizadas
- La arquitectura que se utilizará de cara a la interfaz gráfica y el manejo de los datos por parte de la base de datos.
- Los elementos más importantes de la aplicación de cara a la programación.
- Las soluciones de las partes principales de la aplicación.
- La estructura que va a tener la aplicación.

4.1.Herramientas de trabajo

Para realizar el desarrollo del proyecto utilizaremos un PC, donde se usará el IDE oficial de Android, Android Studio [12] , que al ser multiplataforma también se ha podido realizar desde diferentes sistemas operativos como Mac, Ubuntu y Windows. Android Studio está basado en IntelliJ de JetBrain. Esta herramienta ayuda a agilizar el proceso de desarrollo, y es un IDE gratuito. [13]

Además, para la realización de las maquetas de la interfaz de usuario sobre la estética de la aplicación se ha utilizado la herramienta de código abierto y gratuita Pencil. [14]

4.2.Elementos de la aplicación

Una aplicación Android está basada en una suma de muchos componentes. Todos ellos se conectan entre sí haciendo posible seguir la arquitectura de MVC. Además, se hablaran de otras clases y ficheros importantes en cualquier aplicación Android.

4.2.1.Manifest

Es un archivo xml que está situado en el directorio raíz de la aplicación. Es el encargado de dar la información esencial al dispositivo acerca de qué necesita el sistema para poder ejecutar el código de la aplicación. Parte de los datos importantes que contiene son:

- Contiene el identificador o nombre del paquete de Java para la aplicación.

- Se describen componentes que conforman la aplicación, tales como las actividades y proveedores de contenido. Nombra cada una de las clases de los componentes y publica sus capacidades.

- Determina los procesos que alojan a los componentes de la aplicación.

- Declara los permisos que debe tener la aplicación, ya sea para acceder a partes protegidas de la API o para interactuar con otras aplicaciones. En esta aplicación se necesita permisos de localización.

- Declara la API mínima que debe tener el dispositivo para ejecutar la aplicación. [15]

4.2.2.Actividad

Las actividades (clase Activity) en Android forman parte de la capa vista. Son las clases encargadas de mostrar al usuario los datos que contiene la aplicación. Para conseguir cargar la interfaz, se cargan ficheros xml (Layout). A la vez que se carga esta información, la parte lógica escrita en Java manipula los datos que aparecerán en la pantalla.

La aplicación diseñada para este trabajo contiene muchas actividades, donde para pasar de unas a otras se utilizan intentos (clase Intent) siempre y cuando estén definidas en el manifiesto. Se pueden pasar información entre actividades mediante a esta clase Intent o mediante la clase Bundle.

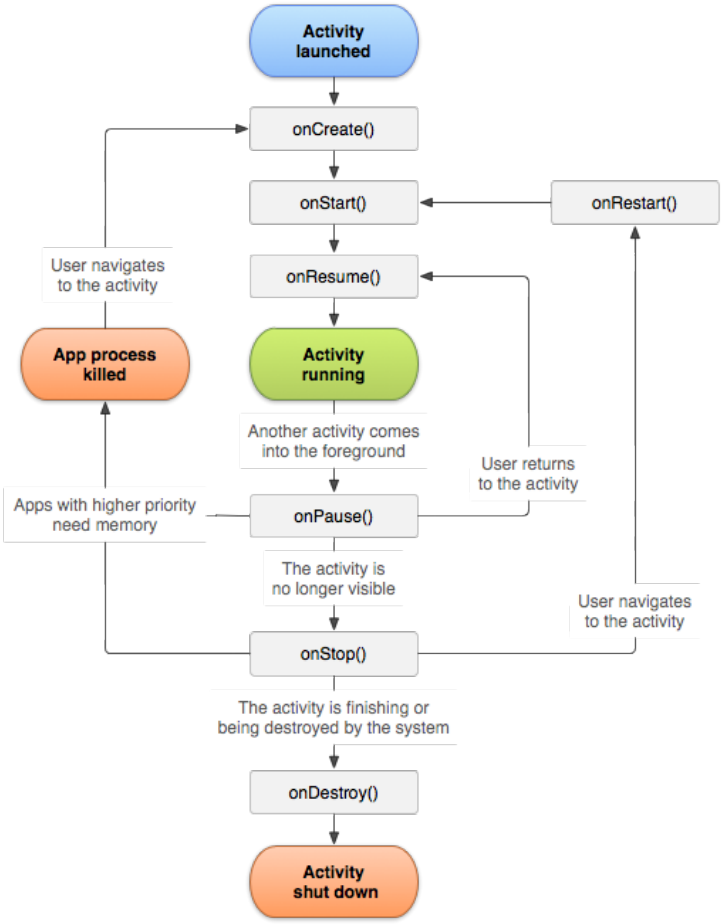

Figura 4-1: Ciclo de vida de una actividad. Recuperado de [16]

Las actividades en una aplicación siguen un ciclo de vida, que se ve afectado por la asociación que tienen con otras actividades [17], tareas y la pila de actividades del dispositivo. Los estados básicos de una actividad son:

- **Reanudada o en ejecución:** es cuando la actividad se encuentra en la pantalla del dispositivo.

- **Pausada:** cuando otra actividad o fragmento está por encima de esta pero no la cubre del todo. En este momento la actividad sigue conteniendo información y está anexado al administrador de ventanas, puesto que sigue en memoria, pero el dispositivo puede eliminar esta actividad si fuera necesario.

- **Detenida:** la actividad está en segundo plano. La diferencia entre detenida y pausada es que la actividad en este estado no está anexada al administrador de ventanas, siendo más probable su eliminación de la memoria.

4.2.3.View

Es la clase que cuya estructura de datos contiene todo lo relativo al objeto de la capa de la interfaz. Es la base de otras clases llamadas widgets o subclases: Text, Button, ImageView, EditText. [18]

Para recoger los eventos de interacción del usuario con la aplicación, las vistas recogen los datos mediante dos métodos: escuchadores y maneadores de eventos

- Los escuchadores de eventos (Event Listener) son una interfaz de View que contienen métodos callback, cada uno único para cada escuchador. Algunos de ellos, utilizados en este trabajo son onClickListener, onLongClickListener u onTouchListener.

- Los maneadores de eventos simplemente son una subclase de la clase View en los cuales se sobrescribe algunas funciones. Esto hace que el Event Listener sea más sencillo, puesto que no se utilizan interfaces ni métodos callback.

4.2.4.Adaptadores

Un adaptador es un objeto View que implementa una interfaz (Adapter). Este objeto actúa como puente el modelo y la vista. Es capaz de mostrar información estructurada en una lista (clase List) o un cursor (clase Cursor) en un widget de View.

En este trabajo se han utilizado adaptadores para el widget RecyclerView en la vista y listas de objetos con las clases Tramo, Sector y Circuito. Para ello se crean unos adaptadores personalizados para cada clase, y en ellos una clase Holder. Una vez agregada una lista de objetos al Adaptador, la clase Holder se encarga de la puesta en la View de la información que contiene las clases de la capa modelo.

Los adaptadores tienen una clase interna, pública y estática que extiende a la clase de RecylcerView que es ViewHolder. Esta clase se encarga de que en cada CardView se

introduzcan los datos a la capa de la vista y funciona también como manejador del método callback onClick.

4.3.Funcionalidades de la aplicación

A continuación se van a explicar las funcionalidades principales de la aplicación. Para ello se mostrará una captura de la interfaz y se analizarán cada uno de sus elementos y la lógica que contienen dichas funcionalidades.

4.3.1.Listar circuitos, sectores y tramos

Es la parte más básica de la aplicación, mostrar un listado de los circuitos que tiene creados un usuario, los sectores asociados a esos circuitos y los tramos que contiene cada uno de los sectores anteriores.

Circuitos

Para la capa de la vista se ha implementado un RelativeLayout. Contiene un título o rotulo del tipo Text, que expone de quién son los circuitos que se están mostrando. Además, el layout contiene un RecyclerView para mostrar de manera más ordenada los datos. Para cada RecyclerView se han creado también otros ficheros xml del tipo CardView.
El layout q contienen un FloatingActionButton con una imagen del tipo Vector con un símbolo de adición (+).

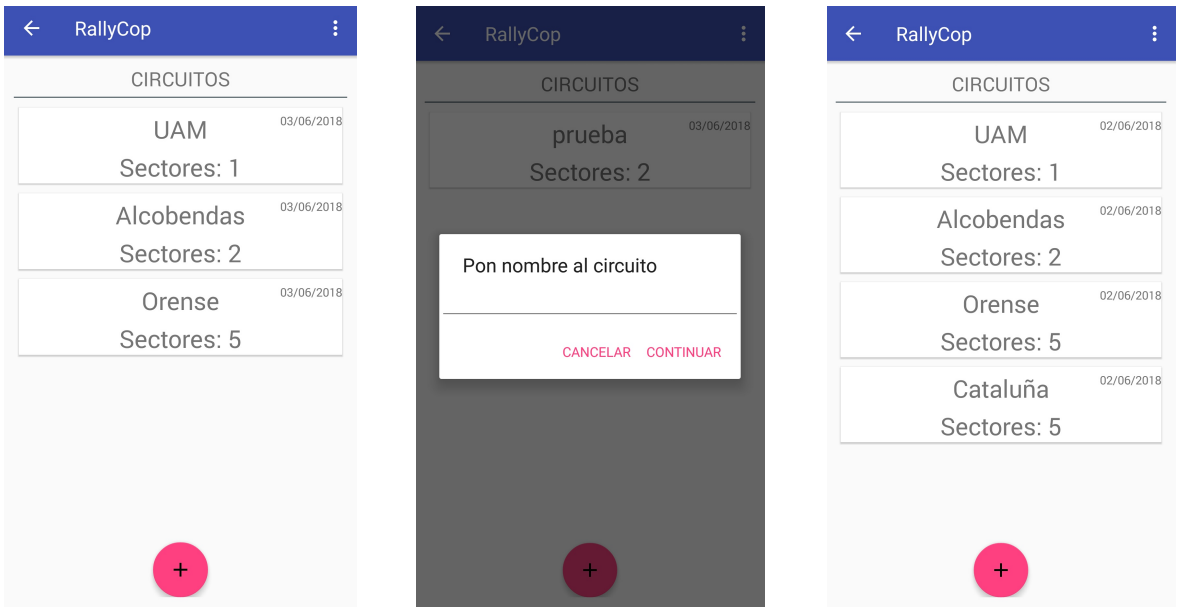

Figura 4-2: Listado de circuitos - añadir circuito

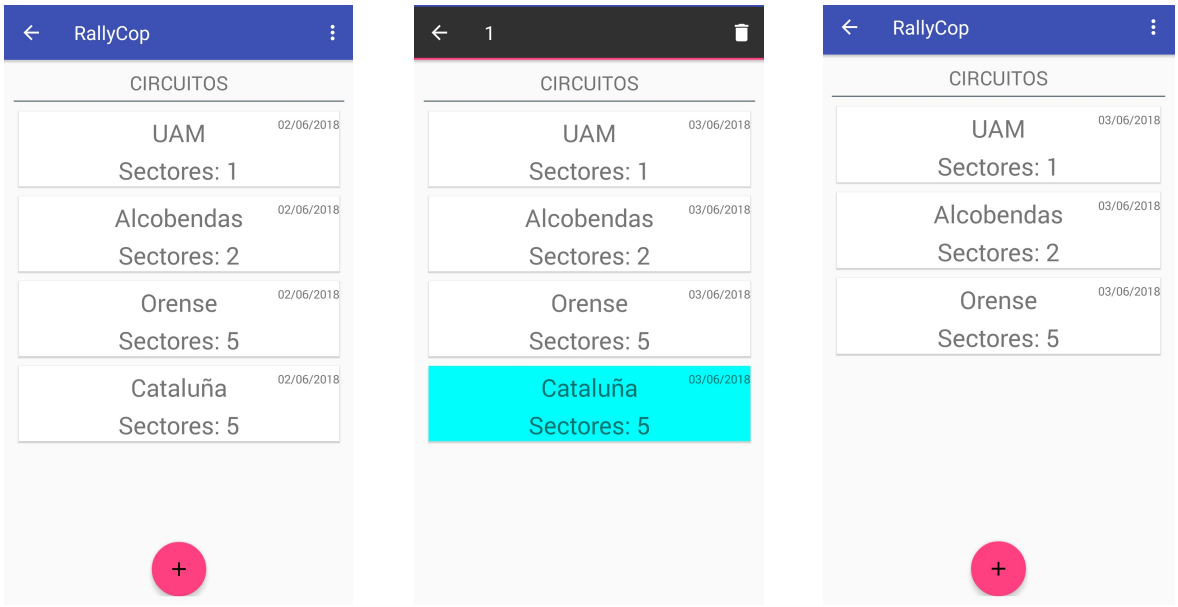

Figura 4-3: Listado de circuitos - borrar circuito

En la parte lógica de la actividad se gestiona el RecyclerView, donde se dota de información gracias a un adaptador y a la interfaz RepositorioCircuitos, que obtiene de la base de datos los circuitos asociados al UUID del usuario. Para añadir los circuitos, se ha asociado un onClickListener al FloatingActionButton. Lo que hace este callback es crear un dialogo invocando a una subclase propia que extiende a la clase DialogFragment, que contiene un EditTex donde introducir el nombre que se le quiere dar al circuito. Después, crea un circuito y lo añade en la base de datos con el nombre introducido. Por último, también se ha añadido un escuchador para los elementos del RecyclerView, donde se ha creado una clase propia que hace que introduzca un onLongClickListener para poder empezar a seleccionar los circuitos que se quieren borrar y un onClickListener cuando se quieran seleccionarás circuitos para borrarlos. Cuando un elemento del listado se pulsa durante mucho tiempo, se crea un ActionMode o menú conceptual para poder eliminar los elementos que se seleccionen de la lista, y cambia el menú de la ActionBar.

Sectores

La capa de la vista se ha implementado muy parecida a la del listado de circuitos, pero con datos diferentes que mostrar.

En la parte lógica de esta actividad también es muy similar a la del listado de circuitos,. No hay que introducirles el nombre, se genera un nombre por defecto y no se guarda en la base de datos.

Tramo

El RelativeLayout que contiene la vista de servicio es muy parecido a los anteriores, salvo por el hecho de que no tiene un FloatingActionButton para añadir circuitos.
Las CardView que contienen la información de los tramos se componen de: un FloatingActionButton con un Vector que contiene la imagen de un lápiz, una ImageView que contiene una flecha con una dirección, varios Text con los datos de la distancia, el

tiempo y la velocidad y un pequeño GridLayout con ImageViews que muestran la información adicional del tramo, si la tuviera.
En la capa de controlador de esta vista, cuando se pulsa durante un largo tiempo un botón de añadir información, es la clase que extiende el ViewHolder quien se encarga del callback onClickListener.

Figura 4-4: Listado de sectores

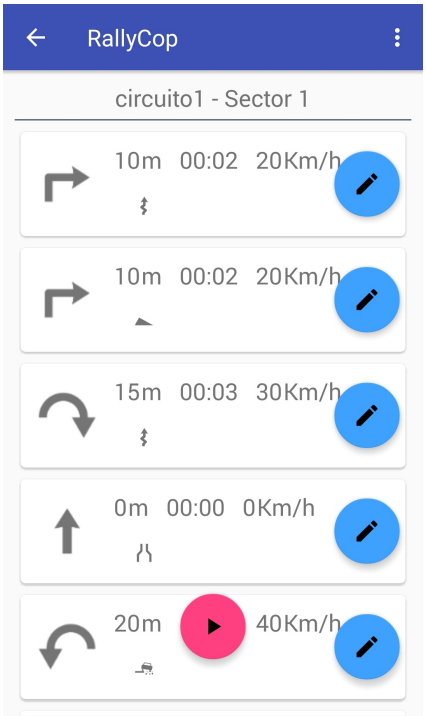

Figura 4-5: Listado de tramos

4.3.2.Registrar tramos

Es el caso de uso principal de esta aplicación. En esta actividad se crean tramos en un sector y se registran los datos de tiempo, distancia y velocidad a partir de coordenadas de latitud, longitud y altura.

En la vista se ha implementado un RelativeLayout que contiene 2 ActionFloatingButton: empezar/terminar y registrar tramo. Además contiene elementos Text que contendrán los valores de distancia y tiempo total del sector y del tramo que se esté añadiendo y analizando coordenadas obtenidas por el localizador.

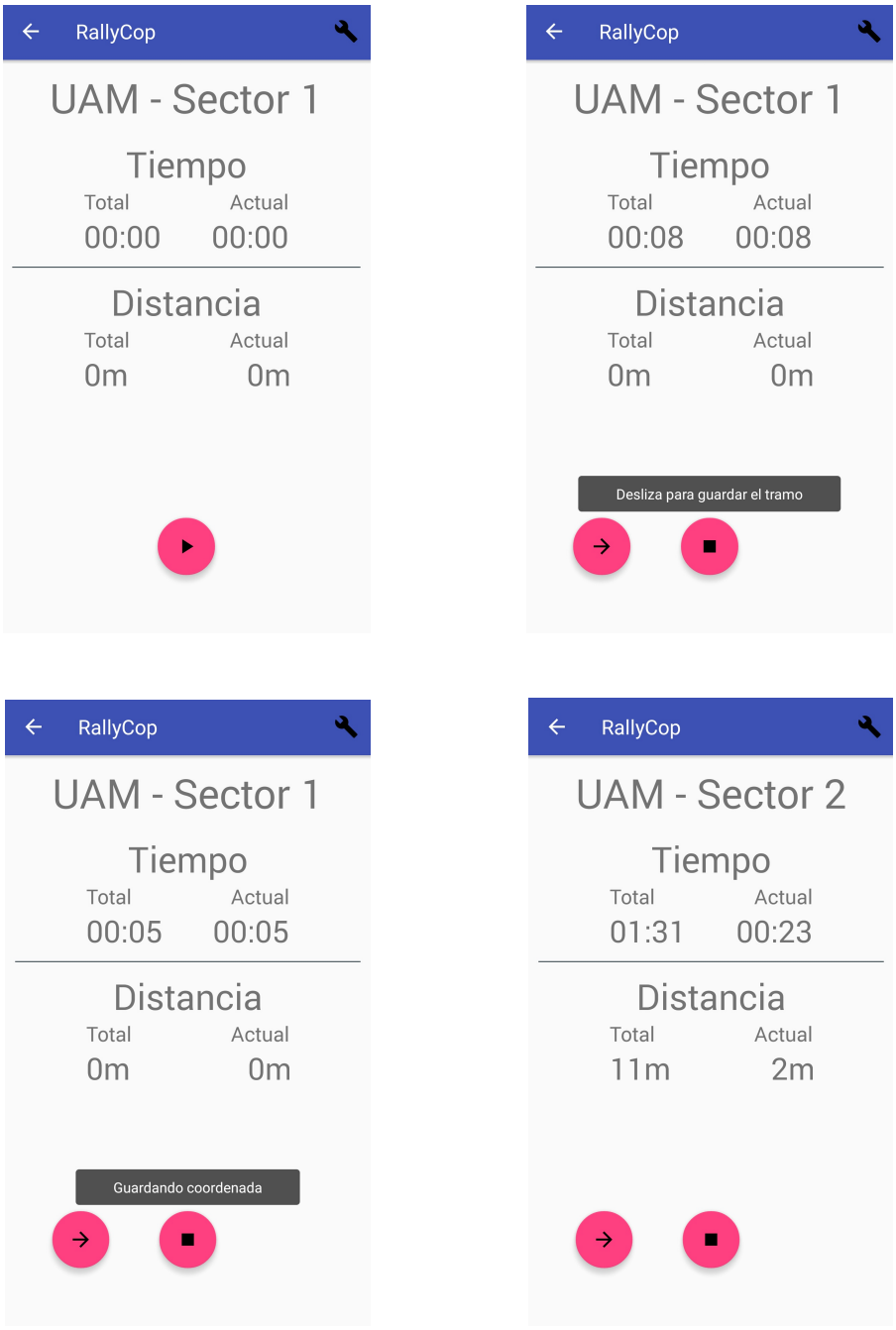

Figura 4-6: Actividad: Registrar tramos

En la capa lógica de esta actividad se encuentran un onTouchListener que se activa cuando se realiza el gesto de arrastrar y solar que tiene el botón de registrar tramo. Este está dividido en un switch con las siguientes opciones que se obtiene de la clase View:

- MotionEvent.ACTION_DOWN: Se trata del momento en el que el usuario pulsa el botón (cuando el botón "baja"). Aquí se recogen los datos de la posición inicial del botón de registrar circuito para más tarde restaurar sus valores de situación en la pantalla.

- MotionEvent.ACTION_MOVE: Aquí simplemente se va actualizando la posición del botón registrando el punto en el cual se encuentra el movimiento.

- MotionEvent.ACTION_UP: Cuando el usuario suelta el botón (cuando se "levanta"), se obtiene la distancia que ha recorrido el botón, y en función de esta se muestra un mensaje que explica que hacer con el botón o se registra el fin de un tramo. Cuando ocurre esto último, se analiza la información recopilada en las coordenadas del tramo para hallar el ángulo de giro si lo hubiera habido y se guarda en la base de datos.

Por otro lado, se encuentra el OnLongClickListener que implementa el botón empezar/terminar. Si no tuviera activado el dispositivo la localización aparece un dialogo que nos lleva a los ajustes en la sección de localización para activar dicha función del dispositivo. Si el botón contiene el símbolo de reproducir se obtiene el proveedor de datos que tiene el usuario en sus preferencias. Una vez hecho esto, se crea un LocationManager que contiene un escuchador (LocationListener) que implementa la actualización de la interfaz en la sección de distancia recorrida. Para calcular esta distancia se cogen las dos últimas coordenadas del tramo y gracias a la función de la clase Location se puede calcular la distancia entre dos puntos. Esta distancia se añade al tramo y se utiliza para calcular la velocidad media.

4.3.3.Añadir información extra

Esta tarea trata de dar más información a un tramo por cosas que mediante coordenadas GPS no se pueden saber, como por ejemplo: el tramo es una zona resbaladiza o está próximo a un acantilado.

Para esta tarea se ha implementado una vista mediante un RelativeLayout. Este contiene un título, compuesto por los nombres del circuito, el sector y el tramo el cual se va a añadir información. Además contiene un GridLayout en el que hay una serie de ImagesViews con imágenes de algunas señales de tráfico. Por último, hay un FloatingActionButton en la vista con la imagen de un disquete.

En la capa lógica de esta actividad se centra en cómo adecuar las imágenes a la pantalla, para que se puedan ver claramente. Además, cada ImageView implementan un Event Listener con un callback onClickListener, que cuando se dispara, la ImageView transforma su fondo a un color cian claro si ha sido seleccionada por primera vez, y lo quita este color

cuando se vuelve a clickar sobre la imagen. Cuando se pulsa sobre el FloatingActionButton, se dispara un método callback onClickListener que guarda en la base de datos el tramo con su información extra modificada.

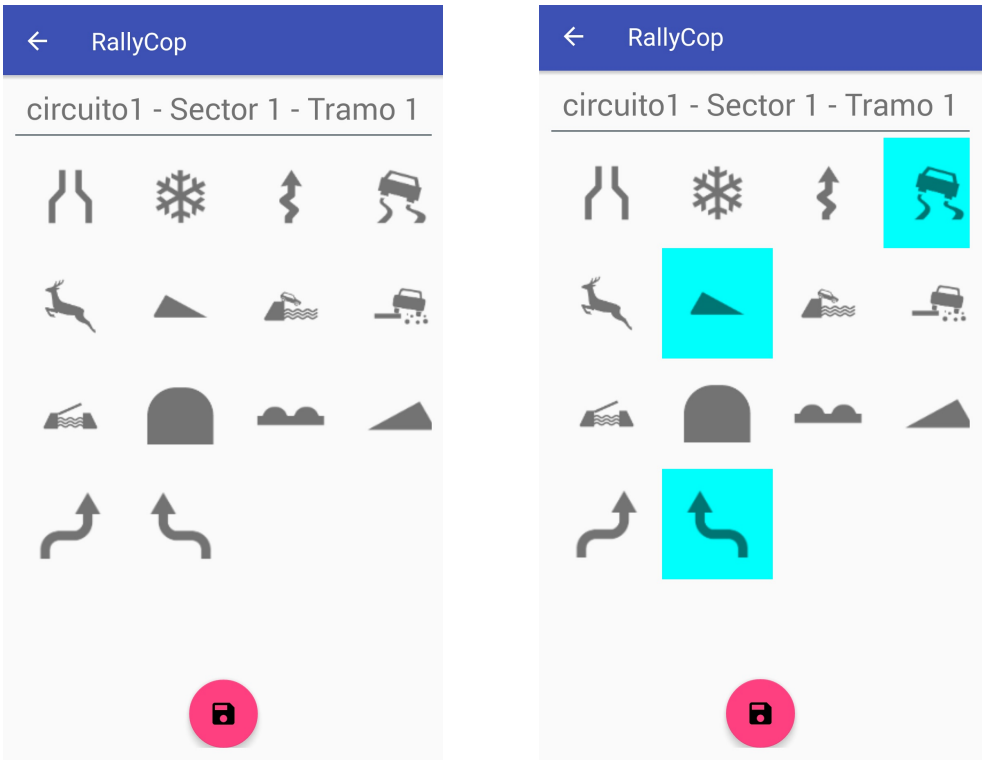

Figura 4-7: Actividad: Añadir Información extra.

4.3.4.Mostrar información en carera

La función de este caso de uso es mostrar al usuario los datos de un sector recogidos previamente y que este pueda comparar sus tiempos con los recogidos previamente.

Para ello se ha realizado una vista mediante un RelativeLayout. Su contenido es de 4 CardView, 3 de ellas con un ImageView con la dirección resultante del tramo analizado, su distancia de tramo, el tiempo en realizarlo, la velocidad media a la que se ha recorrido y un GridLayout con imágenes relativas a la información extra añadida por el usuario. La CardView principal contiene las imágenes más grandes. Los textos relativos a la información del tramo se encuentran encima de las imágenes y los datos que son relativos a las medidas actuales se encuentran debajo de las imágenes. Además, el relativo layout contiene tres FloatingActionButton que contienen imágenes de Reproducir, una flecha que apunta a la derecha y otra que apunta a la izquierda.

En la parte lógica de esta actividad se ha implementado a base de funciones que actualizan constantemente la interfaz. La información relativa a los tramos se recoge desde la base de datos y se guarda en una lista de tramos, donde será más fácil y accesible su información.
Para empezar la reproducción del circuito, se ha designado un onClickListener al botón que contiene la imagen de reproducir, haciendo que el LocationManager empiece a recoger datos para calcular la distancia recorrida actual del usuario y haciendo aparecer los botones

de izquierda y derecha. En ambos botones, izquierda y derecha, se ha implementado un escuchador que realiza un cambio de interfaz.

Si nos encontramos en un punto donde no hay tramos previos o posteriores al tramo actual, sus respectivos botones de desplazamiento y card desaparecen, evitando así poder continuar recorriendo tramos que no existen en la lista de tramos.

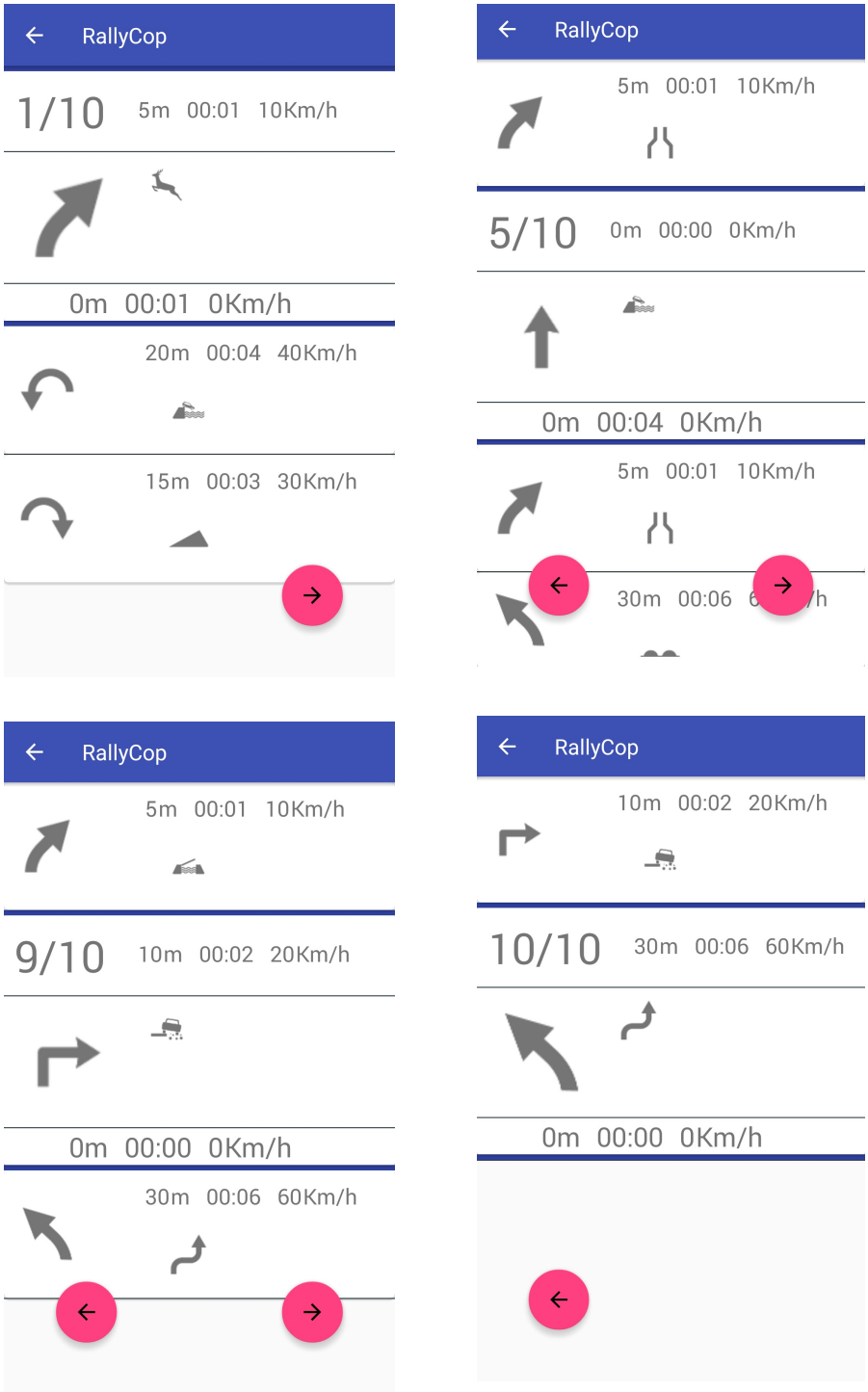

Figura 4-8: Actividad: Mostrar información en carrera.

5.Integración, pruebas y resultados

5.1.Decisiones importantes

En este apartado se mencionaran estrategias y tomas de decisiones que se han hecho en lo que se basa esta aplicación, los tramos de un sector.

Análisis de coordenadas para saber la dirección que sigue el tramo
Un punto vital de esta aplicación es que es capaz de analizar el tramo a partir de las coordenadas GPS que se han recogido durante la actividad de registrar los tramos.

Cuando se registra el final de un tramo y comienza el siguiente se analizan las coordenadas y a partir de la longitud y la latitud de 3 puntos geográficos se puede obtener una medida del ángulo de giro realizado. Si el tramo contiene solo 2 coordenadas se entiende que es una recta y si hubiera más de 3 coordenadas se añade el ángulo de giro de manera recursiva hasta que no quedan coordenadas que registrar.

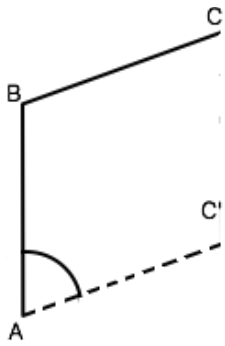

Figura 5-1: Cómo hallar el ángulo entre 2 rectas

La implementación del código que ha hecho a partir de una función que calcula el ángulo entre 3 puntos de Carlos Torres Gutiérrez: [19], transformar una coordenada esférica a una coordenada cartesiana [20] y a transformar este ángulo devuelto en un índice según los grados que se obtenga del análisis de puntos.

Gesto a efectuar para registar un tramo
Para registrar el inicio de recogida de datos se ha utilizado un botón que encienda un temporizador para mostrar los segundos de realización de los tramos y la activación del localizado para guardar las coordenadas del dispositivo. Una vez que se pulsa el botón, aparece el botón con una flecha que indica que se puede desplazar el botón. Si este botón se desplaza más de la mitad de la pantalla se registra el fin de un tramo y el comienzo de uno nuevo en el que guardar los datos.

Se ha escogido este gesto a realizar porque en un coche los toques involuntarios sobre un dispositivo pueden ocurrir con frecuencia y se pretende que el usuario haga un gesto más complejo para poder registrar un tramo.

5.2.Pruebas

En esta sección se han realizado las pruebas para comprobar que los datos que maneja la aplicación, ya sea de manera visual como de manera lógica, individualmente o en conjunto, sean siempre los mismos. Para ello se han implementado pruebas unitarias, pruebas de interfaz y Checkstyle.

5.2.1.Pruebas unitarias

Para la realización de las pruebas unitarias se ha utilizado la librería JUnit de Java. Esto ayuda a realizar test de manera controlada sobre los elementos del modelo que se han implementado durante el desarrollo. A lo largo del desarrollo, se han ido añadiendo pruebas o eliminando ya que el modelo ha tenido más información de la requerida o a sido necesario obtener información más específica, sobre todo en el modelo de Tramo.

En la realización de este apartado se han hecho pruebas de caja negra sobre el modelo. Esto significa que se ha introducido unos valores a cada una de las funciones de los modelos y se ha esperado una cierta salida. Para comprobar que los valores sean correctos se han utilizado funciones Assert (“Equals”, “NotEquals”, “True” y “False”).

5.2.2.Pruebas de interfaz

Para la implementación de las pruebas de interfaz se han utilizado las pruebas Espresso que implementa la herramienta Android Studio. Esta herramienta graba todos los gestos e interacciones que se hacen con la aplicación, permitiendo así un entendimiento más visual de los datos que se quieren comprobar. Además, cuando crea el código utiliza JUnit4 para la realización de los test. Este método permite al programador que se desarrollen pruebas sin necesidad de acceder directamente a las actividades y sus vistas.[21]

Los datos a comprobar de este tipo de pruebas son si existen ciertos elementos en la interfaz o que no aparezcan, o comprobar si ciertos textos escritos se escriben siempre de manera correcta. Se han realizado estas pruebas sobre los casos de uso de Listar los circuitos de un usuario, los sectores de un circuito elegido, los tramos de un sector específico, añadir información extra sobre un tramo y la reproducción de los tramos de un sector.

5.2.3.CheckStyle

Esta herramienta ha sido desarrollada para seguir unos estándares de código Java. El estilo de codificación elegido ha sido el de Google, ya que se requiere para una aplicación siempre y cuando esta se vaya a subir a Google Play. Para el uso de esta herramienta se ha procedido mediante linea de comandos. Primero se obtiene un JAR de la herramienta en formato JAR [22]. Después es necesario obtener los estándares de Google de codificación en formato XML [23]. Por último, se ejecuta la siguiente linea de comandos [24]:

```
java -jar checkstyle-8.10.1-all.jar -c google_checks.xml
<Fichero Java o Carpeta a analizar>
```

Una vez hecho esto, la consola muestra por pantalla los errores y precauciones del código:

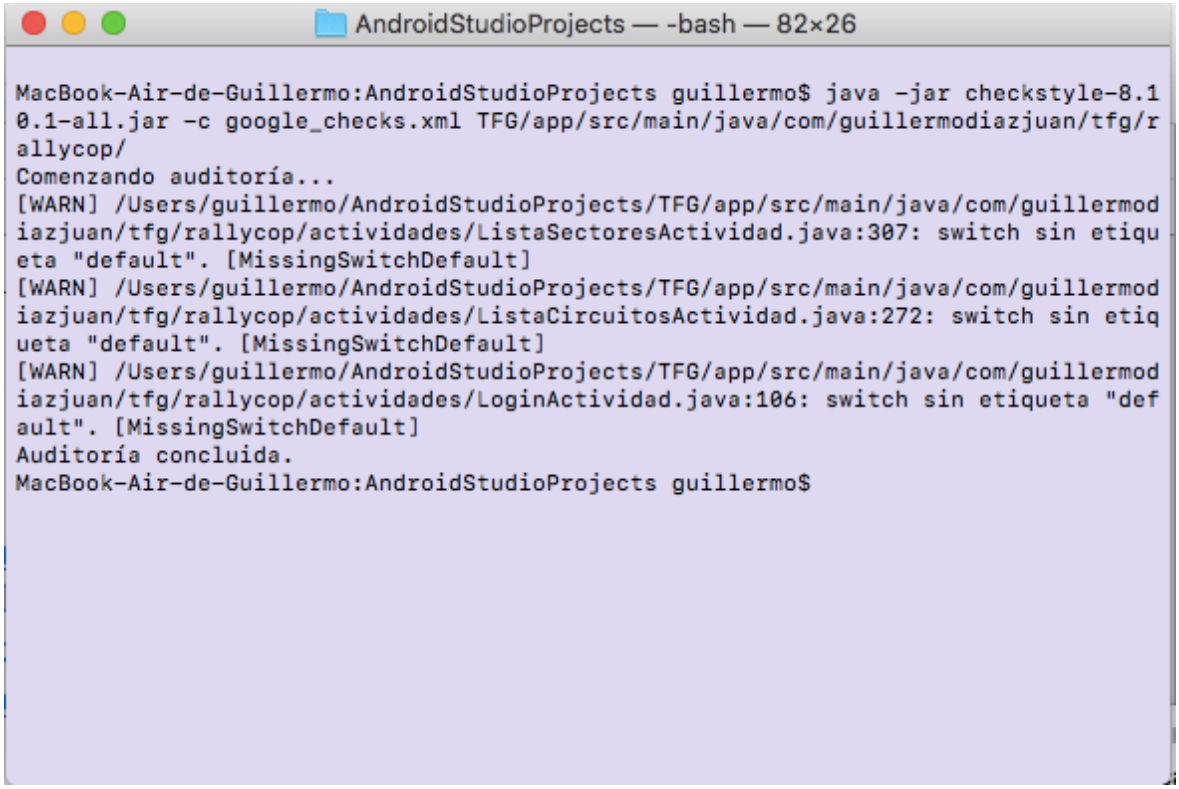

Figura 5-2: Uso de Checkstyle

6.Conclusiones y trabajo futuro

6.1.Conclusiones

En este documento se ha presentado la realización de una aplicación Android de asistencia a pilotos y copilotos en competiciones automovilísticas, especialmente dedicado a la competición de rallies.

En el estado del arte se ha realizado un análisis de varias aplicaciones existentes en el mercado que también tratan la asistencia a pilotos y copilotos de rallies, se ha buscado la captación de datos de una manera más ligera y entendible para el usuario.

Una vez estudiado el estado del arte, se ha realizado un estudio previo sobre Android, y cómo hacer una aplicación para llegar a la mayoría de dispositivos posible. Para ello se ha escogido cómo versión mínima de Android la versión 4.0. Después se ha trazado una metodología en cascada simple. Para realizar una estimación de tiempos al realizar la planificación de desarrollo de la aplicación, se ha procedido al análisis de requisitos tanto funcionales como no funcionales que debe poseer esta aplicación. Una vez se tienen todos los datos del análisis, se ha procedido a la realización de una planificación mediante un diagrama de Gantt para estimar tiempos en la etapa posterior del desarrollo.

Para el desarrollo de la aplicación se ha utilizado el lenguaje de programación Java. Esta etapa se ha realizado siguiendo una arquitectura con patrón de diseño Modelo-Vista-Controlador, que es necesario seguir para seguir los estándares de programación de Google.

Durante la integración ha sido necesario utilizar herramientas de control de versiones y organización de tareas tales como BitBucket y Trello.

En las pruebas realizadas para la aplicación se han utilizado pruebas unitarias y de interfaz utilizando las librerías de Java JUnit, haciendo posible ver posibles fallos de la aplicación ya sea en la capa del modelo o la interfaz. Por último al código se le ha exigido que siga las reglas de codificación de Google para cuando se suba a Google Play pueda hacerse sin que Google extraiga errores o advertencias que no acepte para subir la aplicación.

6.2.Trabajo futuro

Una vez terminado el trabajo, y mediante las pruebas de usabilidad aportadas a varias personas se propone seguir trabajando en los siguientes aspectos:

- Mejorar la interfaz de la aplicación. Se pueden introducir más elementos que hagan más visual los elementos a tratar como las tarjetas de los tramos, sectores y circuitos.

- Ampliar los datos a guardar. Esto da pie al estudio de los usuarios sobre sus rendimientos en carrera y poder saber qué aspectos mejorar de cara a una competición.

- Se podría generar un servidor que guarde carreras de los usuarios o de competiciones específicas, donde con algunos datos relevantes del trazado del circuito, los competidores puedan simplemente añadir sus tiempos realizados.

- Análisis en diferentes etapas de una competición de rallies. Sería interesante elegir que etapa se está realizando para el análisis de datos, para después realizar una estimación en base a una información acerca del coche que utilicen los usuarios.

Referencias

1. FiA. World Rally Championship. Regulaciones y apéndices de 2018 [archivo PDF]. Recuperado de https://www.fia.com/file/66878/download/9275?token=82ABWjwm. p 11. Último acceso 17 de junio de 2018.
2. Daniel Cárdenas. Ventajas y desventajas de programar en JAVA. http://adictoalcodigo.blogspot.com/2016/07/ventajas-y-desventajas-de-programar-en.html. Último acceso 18 de junio de 2018.
3. Versiones android: https://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android. Último acceso 18 de junio de 2018.
4. Fundamentos de Sistemas de Información 201. Modelos del ciclo de vida de software: http://fsi201lsca.blogspot.com.es/2011/02/modelos-del-ciclo-de-vida-de-software.html. Último acceso 18 de junio de 2018.
5. Imagen ciclo de cascada: http://3.bp.blogspot.com/-VCq5TNHp5HY/UWzPqIr2JYI/AAAAAAAAAAU/LJ7QSJnMOCk/s1600/modelo-en-cascada.png. Último acceso 18 de junio de 2018.
6. Jose Angel Zamora. Cómo testar tu aplicación en Android: https://elandroidelibre.elespanol.com/2016/03/como-testear-tu-aplicacion-en-android.html, marzo de 2016. Último acceso 18 de junio de 2018.
7. Descripción de ArgoUML. http://argouml-stats.tigris.org/documentation/quickguide-0.34-single/quickguide.html. Último acceso 18 de junio de 2018.
8. Versiones android y tamaño de dispositivos: https://developer.android.com/about/dashboards/index.html. Último acceso 18 de junio de 2018.
9. Cristian Henao. Ejemplo Modelo Vista Controlador. http://codejavu.blogspot.com/2013/06/ejemplo-modelo-vista-controlador.html. Junio de 2013. Último acceso junio de 2018.
10. Imagen de arquitectura Modelo-Vista-Controlador: https://upload.wikimedia.org/wikipedia/commons/a/a9/ModelViewControllerDiagram_es.svg. Último acceso 18 junio de 2018.
11. Cristian Henao. Ejemplo de Modelo-Vista-Controlador: http://codejavu.blogspot.com/2013/06/ejemplo-modelo-vista-controlador.html. Junio de 2013. Último acceso 18 de junio de 2018.
12. AndroidStudio. https://developer.android.com/studio/intro/index.html?hl=es-419. Último acceso 18 de junio de 2018.
13. A. Gerber and C. Craig, Learn Android Studio: Build Android Apps Quickly and Effectively, Apress, capitulo 1.
14. Descripción de Pencil: http://pencil.evolus.vn. Último acceso 18 de junio de 2018.
15. Archivo Manifest en Android: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=ES. Último acceso 18 de junio de 2018.
16. Imagen ciclo de vida de una actividad Android: https://developer.android.com/images/activity_lifecycle.png?hl=es-419. Último acceso 18 de junio de 2018.

17. Estados básicos de una actividad Android. https://developer.android.com/guide/components/activities?hl=es-419. Último acceso 18 de junio de 2018.
18. Interfaces de usuario. https://sites.google.com/site/swcuc3m/home/android/api/librerias-basicas-interfaces-usuario. Último acceso 18 de junio de 2018.
19. Carlos Torres Gutiérrez. Cálculo del Ángulo implementado en Java. http://www.dccia.ua.es/dccia/inf/asignaturas/RG/2002/material/RecConvexo2D/web2/imple_angulo.html. Último acceso 18 de junio de 2018.
20. rbrundritt. Conversion between Spherical and Cartesian Coordinates Systems. https://rbrundritt.wordpress.com/2008/10/14/conversion-between-spherical-and-cartesian-coordinates-systems/. Último acceso 19 de junio de 2018.
21. Pruebas Espresso en Android: https://developer.android.com/studio/test/espresso-test-recorder?hl=es-419. Último acceso 18 de junio de 2018.
22. Archivo JAR para realizar Checkstyle. https://github.com/checkstyle/checkstyle/releases/download/checkstyle-8.10.1/checkstyle-8.10.1-all.jar. Último acceso 18 de junio de 2018.
23. Archivo XML de corrección estilo de Google. https://raw.githubusercontent.com/checkstyle/checkstyle/master/src/main/resources/google_checks.xml. Último acceso 18 junio de 2018.
24. Uso de Checkstyle mediante linea de comandos. http://checkstyle.sourceforge.net/cmdline.html. Último acceso 18 de junio de 2018.

Glosario

GPS	Sistema de posicionamiento global
TFG	Trabajo final de grado
API	Interfaz de programación de aplicaciones
SO	Sistema operativo
UML	Lenguaje unificado de modelado
RF	Requisito funcional
RNF	Requisito no funcional
MVC	Modelo-Vista-Controlador
VO	Objeto de valor
SQL	Lenguaje de consulta estructurada
PC	Computadora Personal
IDE	Entorno de desarrollo integrado